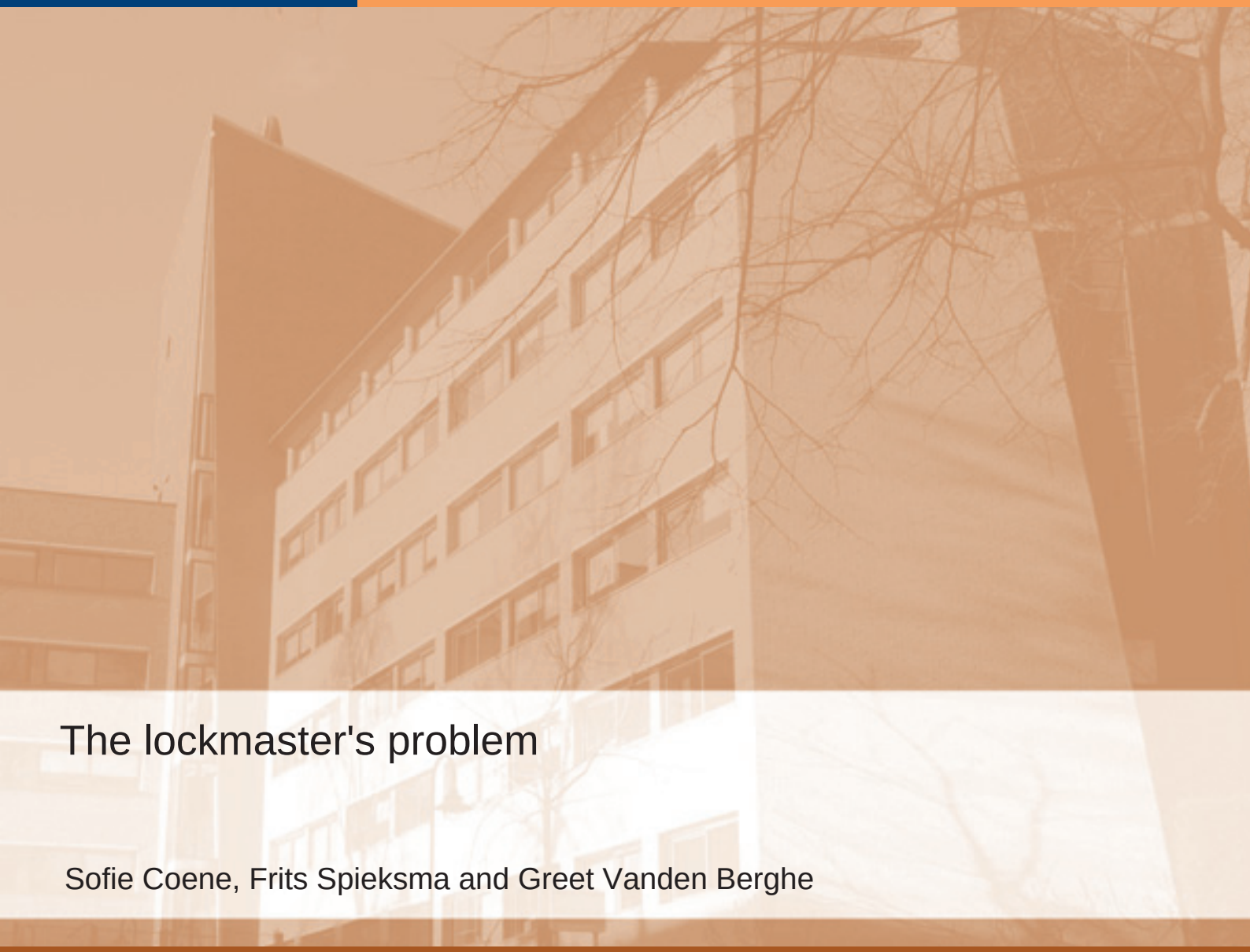
KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics



The lockmaster's problem

Sofie Coene, Frits Spieksma and Greet Vanden Berghe

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 1126

The Lockmaster's Problem*

Sofie Coene† Frits C.R. Spieksma†
Greet Vanden Berghe ‡

September 30, 2011

Abstract

Inland waterways form a natural network that is an existing, congestion free infrastructure with capacity for more traffic.Transportation of goods by ship is widely promoted as it is a reliable, efficient and environmental friendly way of transport. A bottleneck for transportation over water are the locks that manage the water level. The lockmaster's problem concerns the optimal strategy for operating such a lock. In the lockmaster's problem we are given a lock, a set of ships coming from downstream that want to go upstream, and another set of ships coming from upstream that want to go downstream. We are given the arrival times of the ships and a constant lockage time; the goal is to minimize total waiting time of the ships. In this paper a dynamic programming algorithm (DP) is proposed that solves the lockmaster's problem in polynomial time. We extend this DP to different generalizations that consider weights, water usage, capacity, and (a fixed number of) multiple chambers. Finally, we prove that the problem becomes strongly NP-hard when the number of chambers is part of the input.
Keywords: lock scheduling, batch scheduling, dynamic programming, complexity

1 Introduction

Transportation of goods by ship, over sea as well as over waterways, is a promising alternative for transport over land. Reasons are its reliability, its

*An extended abstract of a preliminary version of this work has been accepted for ATMOS 2011, Saarbruecken

†Research group Operations Research and Business Statistics (ORSTAT), Katholieke Universiteit Leuven, Belgium

‡Information Technology, KaHo Sint-Lieven, Gent, Belgium

efficiency (a ship of 1200 tons can transport as much as 40 train couches and 60 trucks), and its environmental friendliness. Here, we focus on transport by inland ships over waterways. For instance, the European Commission promotes the better use of inland waterways in order to relieve heavy congested transport corridors. Carriage of goods by inland waterways is a mode of transport which can make a significant contribution to sustainable mobility in Europe [7, 1]. Not only is its energy consumption per km/ton of transported goods approximately 17% of that of road transport and 50% of rail transport, it also has a high degree of safety and its noise and gas emissions are modest. This natural network is the only existing infrastructure that is congestion free and has capacity for more traffic [9]. In the US, total waterborne commerce has risen from about 1,500 million tons of goods in 1970 up to 2,600 million tons in 2006; due to the economic crisis it has lowered since then to a level of about 2,200 million tons in 2009 [5].

Typically, these waterways are interrupted by locks that are able to maintain higher water levels such that larger and heavier ships are able to use it. These locks are a bottleneck for transportation over water and hence, operating locks wisely contributes to the popularity of transportation over water. However, the algorithmic problem how to operate a lock has not been studied broadly in the scientific literature. The purpose of this paper is to fill this gap.

We now continue with the description of a very basic situation that will act as our core problem: the lockmaster's problem. Later, we will discuss extensions to more realistic settings. Consider a lock consisting of a single chamber. Ships coming from upstream, wanting to go downstream, arrive at the lock at given times r_i, $i = 1, \ldots, n_1$ with $r_1 \leq r_2 \leq \ldots \leq r_{n_1}$. Other ships, coming from downstream, wanting to go upstream, arrive at the lock at given times s_i, $i = 1, \ldots, n_2$ with $s_1 \leq s_2 \leq \ldots \leq s_{n_2}$. Let $n = n_1 + n_2$, and let T denote the *lockage duration*: this is the time between closing the lock for entering ships, and opening the lock so that ships can leave. We assume that all data are integral. Our goal is to find a feasible lock-strategy that minimizes total waiting time of all ships. In other words, we need to determine at which moments in time the lock should start to go up (meaning at which moments in time ships that are downstream are lifted), and at which moments in time the lock should start to go down (meaning at which moments in time ships that are up are being lowered). Clearly, for such a strategy to be feasible, (i) going-up moments and going-down moments (referred to as moments) should alternate, and (ii) consecutive moments should be at least T time-units apart. It is clear that this particular problem is a simplified version of reality; we will, however, add capacity restrictions and

other extensions in Section 4.

2 Literature

Literature on lock scheduling problems is rather limited. Some recent papers deal with the optimal sequencing for locking ships when a queue emerges due to some lock malfunction or accident. Nauss [14] determines an optimal sequence in the presence of setup times and non-uniform lockage processing times. Smith et al. [17] perform a simulation study on the impact of alternative decision rules and infrastructures improvement on traffic congestion in a section of the Upper Mississippi River. Ting and Schonfeld [18] study several control alternatives, such as sequencing, in order to improve lock service quality. They use heuristic methods. Verstichel and Vanden Berghe [19] mention the increasing occupation of logistic infrastructure in ports and waterways. They develop (meta)heuristics for a lock scheduling problem where a lock has at least one chamber, but often consists of multiple parallel chambers of different dimensions and lockage times. They deal with capacity restrictions in the sense that ships have sizes and the lock area is restricted, making this problem at least as hard as a bin packing problem. None of these papers study the lockmaster's problem.

The lockmaster's problem is closely related to a batch scheduling problem. Batch scheduling involves a machine that can process multiple jobs simultaneously. As far as we are aware, this connection has not been observed so far. Suppose for the moment that, in our problem, we only have downstream going ships. Then, the lock can be seen as a batching machine and the jobs are the arriving ships with release dates and equal processing times (i.e. the lockage time T). Following the notation of Baptiste [2] this is problem $1|p-batch, b=n, r_i, p_i=p|\sum w_i F_i$. In words: we have a single parallel batching machine with unrestricted capacity ($b=n$), release dates on the jobs, and uniform processing times. The objective is to minimize the sum of weighted flow times, however, in the basic lockmaster's problem there are only unit weights. Baptiste [2] shows that this problem is polynomially solvable for a variety of objective functions. Cheng et al. [6] developed an $O(n^3)$ algorithm for $1|p-batch, b=n, r_i, p_i=p|f$ where f can be any regular objective function. Condotta et al. [8] show that feasibility of the same problem with deadlines can be checked in $O(n^2)$, even for a setting with parallel batching machines; in our setting this translates to multiple parallel chambers (see Section 5). Ng et al. [15] study a single machine serial batching scheduling problem with release dates and identical

processing times. Machine setup only happens after arrival of the final job in a batch and there is a fixed setup time equal to s. The completion time of a batch is equal to the sum of the processing times of the jobs in the batch. This problem is equivalent to the uni-directional lockmaster's problem with $s := T$ and $p_i = p = 0$, and can be solved in $O(n^5)$ by a dynamic programming algorithm described in [15]. Clearly, our lockmaster's problem is more general. Indeed, when there are upstream going and downstream going ships, we are dealing with two families of jobs, and only jobs of the same family can be together in a batch. Further, in our case, processing a batch of one family needs to be alternated by processing a (possibly empty) batch containing jobs of the other family; i.e. it is not possible to process two batches of the same family consecutively. The concept of a "family" of jobs is also described by Webster and Baker [20], however not in combination with a batch processing machine. In their paper, Webster and Baker deal with a scheduling problem where scheduling jobs of the same family consecutively reduces setup times. In our problem, dealing with jobs of the same family consecutively, i.e. in one batch, reduces the total batching time. This type of problem is also known under the name of batch scheduling with job compatibilities. Jobs within a batch need to be pairwise compatible, and these compatibilities can be expressed using a compatibility graph. Boudhar [3] and Finke et al. [10] study different variants of these batch scheduling problems when the compatibility graph is bipartite or an interval graph. In our case the compatibility graph is the union of two cliques. Our problem can be summarized as being $1|p - batch, b = n, r_i, p_i = p, \Phi = 2, s_{fg}|\sum F_i$, with $s_{fg} = 2T$ if $f = g$ and $s_{fg} = T$ if $f \neq g$, where Φ refers to the number of families and s_{fg} to the setup times between batches; we will refer to our problem as the lockmaster's problem. For a review on scheduling a batching machine we refer the reader to Potts and Kovalyov [16] and Brucker et al. [4]. A related problem is studied by Lee et al. [13] who develop dynamic programming algorithms for scheduling a batching machine with release dates, deadlines, and constant processing times when the goal is to minimize makespan or minimize the number of tardy jobs. In conclusion, this literature study reveals that the complexity of our lockmaster's problem does not follow from results in literature. Further, we also consider the lockmaster's problem with multiple parallel chambers.

2.1 Our results

We show that
(1) there is an $O(n^5)$ algorithm for the lockmaster's problem (see Section 3);

(2) this algorithm can be extended to deal with regular objective functions (4.1), non-uniform lockage times (4.2), settings with a limited number of times that there can be locked (4.3), capacities (4.4), and with a constant number of parallel chambers (4.5);
(3) if the number of parallel chambers is part of the input, the problem becomes strongly NP-hard (Section 5).

3 A dynamic program for the lockmaster's problem

In this section we describe a dynamic programming algorithm (DP) for our problem. This DP is based on two properties valid for the lockmaster's problem. Informally stated, one property is that when during some time period no ships arrive, we can split the instance into two subinstances (see Lemma 3.1); the other property is that we can restrict the moments in time that need to be considered in an optimal strategy (see Lemma 3.2). We now proceed by formalizing these observations.

Lemma 3.1 *When, for a given instance of the lockmaster's problem, during a time period equal to $4T$ no ships arrive at the lock, the instance can be divided into two instances. The solution can then be found by solving these two smaller instances.*

Proof. Consider an instance I featuring a ship arriving at time $s \in S$, no ships arriving during $(s, s + 4T]$, and at least one ship arriving after $s+4T$. Let I_1 be the subinstance that consists of all ship arrivals until (and including) s, and let I_2 consist of all ship arrivals after $s+4T$. Consider now an optimal solution to I_1. The latest possible optimal lockage time for the ship arriving at s is $s + 2T - \epsilon$, for some (small) $\epsilon > 0$. Indeed, any solution where this ship is locked at time $t \geq s+2T$ is dominated by a solution where the ship is locked at time $t - 2kT$ with k an integer such that $t - 2kT$ is in $[s, s + 2T)$. Thus all ships in I_1 are served before $s + 3T$ (including the T time units needed to transfer the ship arriving at s). Consider now an optimal solution to I_2. This optimal solution has the lock in some position at time $s+4T$, the earliest possible arrival time of a ship in I_2. Since we can already bring the lock in this position starting at time $s+3T$ and since this does not take up more than T time units, it follows that we can concatenate an optimal solution to I_1 and an optimal solution to I_2 in order to find a feasible, and optimal solution to I. □
It follows that, without loss of generality, we can restrict the analysis to

instances of the lockmaster's problem that have the property that a ship arrives during any $4T$ interval.

Now, when is a lock likely to start going up or down? Either upon arrival of a ship or immediately upon arrival of the lock. This suggests that the number of moments the lock starts moving is limited. Garey et al. [12] and recently Condotta et al. [8] use the concept of "forbidden regions" in the presence of deadlines to define periods of time in which no job/batch can start in a feasible schedule. Given that there are no deadlines, the same concept can be used to define periods of time in which no batch can start in an optimal schedule. We introduce a set of moments U at which it is possible to go up. These upmoments are referred to as u_i. Similarly, we introduce a set of moments at which it is possible to go down, the set D. These downmoments are referred to as d_i. Let us define set $S = \{s_i\}$, set $R = \{r_j\}$ and the set $\Theta = \{0, 2T, 4T, \ldots, 4nT\}$; the cardinality of Θ follows from Lemma 3.1. We use the Minkowski-sum to sum two sets, i.e. the sum of two sets $A = \{a_i\}$ and $B = \{b_j\}$ as follows:

$$A + B = \{a_i + b_j | a_i \in A, b_j \in B\}.$$

Then, bearing this definition in mind, here is a proposal for U and for D:

$$U = (S + \Theta) \cup (R + \Theta - \{T\}),$$

$$D = (R + \Theta) \cup (S + \Theta - \{T\}).$$

For example, suppose we have two ships traveling downstream and two ships traveling upstream with $R = \{1, 7\}$ and $S = \{2, 4\}$ and $T = 5$. Then,
$U = \{-4, 2, 4, 6, 12, 14, 16, 22, 24, 26, \ldots, 76, 82, 84\}$ and
$D = \{-3, -1, 1, 7, 9, 11, 17, 19, 21, 27, 29, \ldots, 79, 81, 87\}$.

Notice that the assumption we made based on Lemma 3.1, together with the choice of Θ, ensures that no ship can arrive at a moment not in $U \cup D$. Moreover, it follows easily from the construction of U and D, and the fact that S, R, and Θ contain $O(n)$ elements, that there are at most $O(n^2)$ elements in $U \cup D$.

Lemma 3.2 *There is an optimal lock strategy for the lockmaster's problem whose upmoments are contained in U, and whose downmoments are contained in D.*

Proof. Contradiction. Suppose there is an instance such that each optimal strategy has either an upmoment not in U or a downmoment not in D (such a moment is called a failure). Consider an optimal strategy for this instance

for which its earliest failure is minimal, say at time t. Let us assume for convenience that at time t, the lock went up. Notice that t cannot be equal to an s_i. Consider that moment in time t. Let $\epsilon > 0$ be a very small quantity. There are two possibilities:

(i) at $t - \epsilon$ the lock was waiting to go up. If, in our optimal strategy, there are ships transported up at time t, it cannot have been optimal to wait until t, since no downstream ships arrive at time t (since t is not in S). Hence, there are no ships transported. But then, we need not have waited, and there is an optimal strategy where the lock went up immediately after the last time before t the lock went down.

(ii) at $t - \epsilon$ the lock was going down. Thus, at $t - T$, the lock started a down-operation. This moment in time is, by assumption, in D. But then it follows that t is in U. Contradiction.

□

We now define a dynamic programming algorithm (DP) where $f(u_i, d_j)$ (with $u_i \leq d_j - T$) represents the minimal costs of a lockage strategy that takes care of all up-requests up to u_i, all down-requests up to d_j, which features an upmoment at time $t = u_i$, which features a downmoment at time $t = d_j$, and such that there are no other up- or downmoments in between u_i and d_j.

Here is a recursion. For each $u_i \in U$, $d_j \in D$, with $u_i \leq d_j - T$ we have:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\};$$

for all $u_i > d_j - T$ we set:

$$f(u_i, d_j) = \infty.$$

The recursion is initialized as follows:

$$f(u_1, u_1 + T) = 0.$$

Notice that, according to the definition of U, $u_1 = \min\{s_1, r_1 - T\}$. The optimal value is given by $\min\{f(u_i, d_j) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$. A straightforward way to determine the complexity of DP is to observe that there are $O(n^4)$ states and, since for each state we enumerate over all other states, we arrive at an $O(n^8)$ algorithm.
We now proceed by describing a more careful analysis, leading to an $O(n^5)$ implementation of DP. We observe the following.

Observation 3.3 *If $d_j \in D \setminus R$, then the previous upmoment was $d_j - T$.*

Indeed, notice that if the lock goes down at a moment in time (say t) that is not an arrival moment in R, then the previous upmoment was at $t - T$. If the lock went up earlier than $t - T$, then there is an optimal solution in which the next downmoment is earlier than t; as no ship is arriving at t, there is no need to wait until t.
For a proof of correctness of DP, we refer to the Appendix; here we argue that an $O(n^5)$ implementation of DP is possible.

Theorem 3.4 *DP is a polynomial-time ($O(n^5)$) algorithm for the lockmaster's problem.*

Proof. We argue that Observation 3.3 above implies that it is sufficient for DP to consider $O(n^3)$ states. Indeed, there are $O(n^2)$ states with $d_j \in D \setminus R$, and $O(n^3)$ states with $d_j \in R$. The latter fact follows from the insight that $|R| = O(n)$ (combined with the fact that U and D have cardinality $O(n^2)$). Computing the value of each state can be done by evaluating $O(n^3)$ states, leading to a total time complexity for DP equal to $O(n^6)$. We can, however, speed this up by $O(n)$. First, consider the states with $d_j \in D \setminus R$, such a state can be computed in $O(n^3)$. Computing all these states thus takes $O(n^5)$. Second, consider the states with $d_j \in R$. The computing time for such a state depends on the value of u_i. If $u_i \in S$, the computing time is also $O(n^3)$ for a total of $O(n^2)$ states, yielding a total time complexity of $O(n^5)$. If $u_i \in U \setminus S$, we know that $d'_j = u_i - T$ (due to observation 3.3). It follows that computing each of these $O(n^3)$ states only requires $O(n^2)$ time, again yielding a total time complexity equal to $O(n^5)$. Hence, the total time complexity for this algorithm is $O(n^5)$. □

4 Extensions

4.1 Regular objective functions

For the analysis above we chose as an objective to minimize the sum of the waiting times, which is a very natural objective function for this problem. The algorithm, however, works for any regular, i.e. non-decreasing in (waiting) time, objective function. Such a function can be for instance minimizing the weighted sum of waiting times or minimizing the maximum waiting time. Indeed, in the recursion, a cost of a state can be computed by taking the cost of a previous state and adding the extra cost incurred. These are cost-functions non-decreasing in t and it is clear how the extra

cost can be calculated, independent of the value of the previous state. Let us consider, for example, the weighted lockmaster's problem. In practice, it happens that not all ships are of equal importance, e.g. it is conceivable that the waiting cost for ships transporting goods is higher than the waiting cost of leisure ships; or ships transporting dangerous goods get priority over normal cargo ships, see e.g. [17, 19]. This can be dealt with by assigning weights to the ships revealing their priority. Taking into account weights w for the ships in the DP recursion can be done straightforwardly as follows:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} w_\ell (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} w_k (d_j - r_k)\}.$$

Initialization and determination of the optimal value are identical to the basic DP in the previous section.

4.2 Non-uniform lockage times

It is not uncommon that lockage times for going up (T_u) and down (T_d) differ. Then, for $u_i \in U$ and $d_j \in D$:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T_d \\ u_{i'} \leq d_{j'} - T_u}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\}$$

where Θ, U and D are now:

$$\Theta = \{0, T_d + T_u, 2(T_d + T_u), 3(T_d + T_u), \ldots, n(2T_d + 2T_n)\},$$

$$U = (S + \Theta) \cup (R - \{T_u\} + \Theta),$$

$$D = (R + \Theta) \cup (S - \{T_d\} + \Theta).$$

It is not difficult to verify that all results from Section 3, mutatis mutandis, apply to this setting.

4.3 Water usage

Due to organizational/environmental reasons, there can be a limit on the number of times lockage is allowed in some time-interval. In particular, when water is scarce (e.g. in dry seasons), going into lockage too often results in disruptions of the water level, see [19]. Then, Lemma 3.1 no longer holds.

Indeed, splitting an instance would also mean dividing the number of allowed lockage times over the two instances and it is not straightforward how this should be done. The cardinality of U and D needs to be reconsidered. Let us define alternative sets U' and D' as follows. For all pairs of consecutive ships (t, t') with $m_{t'} - m_t \geq 4T$ and $m_t, m_{t'} \in S \cup R$, let $U' = U \backslash \{u_i | u_i \in [m_t + 4T, m_{t'})\}$ and $D' = D \backslash \{d_j | d_j \in [m_t + 4T, m_{t'})\}$. Then, the following holds.

Lemma 4.1 *There is an optimal lock strategy for the lockmaster's problem with a bound on the number of lockage times whose up- and downmoments are contained in U' and D' respectively.*

Proof. From Lemma 3.2 we know that there is a strategy where all optimal up- and downmoments are contained in U and D. Suppose a ship arrives at time m_t and during a time period of $4T$ after that no other ships arrive. Suppose further, without loss in generality, that the ship arriving at m_t is an upstream going ship. Then, following the same argument as in the proof of Lemma 3.1, all u_i and d_j later than $m_t + 4T$ and earlier than $m_{t'}$, the first arrival after m_t, will not be part of an optimal solution and can be deleted from the sets U and D. □

What is now the cardinality of U' and D'? When, in an instance, every $4T$ time units at least one ship arrives, cardinality is $O(n^2)$ (see Section 3). If this is not the case, it means that there are a number of gaps, let's say x, between two consecutive arrivals with interarrival time larger than $4T$. It holds that $0 \leq x \leq n$, yielding size $O(n^3)$ for $U \cup D$.

In a dynamic programming recursion for this problem (DPw), an entry is needed to keep track of the number of times there has been locked before. It still holds that all ships arrived before or upon lockage time will be handled. Now, we use v for the number of times there has already been locked and V for the maximum number of times there can be locked. For $u_i \in U'$, $d_j \in D'$,$v \leq V$, the algorithm DPw is given by:

$$f(u_i, d_j, v) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}, v-2) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\}.$$

The initial state is

$$f(u_1, u_1 + T, 1) = 0$$

with $u_1 = \min\{s_1, r_1 - T\}$.

The optimum is given by $\min\{f(u_i, d_j, v | u_i \geq s_{n_2}, d_j \geq r_{n_1}, v \leq V)\}$. In

these states all ships are locked and the maximum number of allowed lockage times is not exceeded.

Lemma 4.2 *DPw is a polynomial-time algorithm for the water-usage constrained lockmaster's problem.*

Proof. See also the proof of Theorem 3.4. There are $O(Vn^4)$ states, with $V \leq n$. Computing each state can be done by evaluating $O(n^4)$ states, leading to a total time complexity for the algorithm equal to $O(Vn^8)$. This can be further reduced by applying similar arguments as in the proof of Theorem 3.4. □

4.4 Capacity

Until now we did not take into account any capacity restrictions. We will discuss two different approaches dealing with capacity restrictions in this subsection.

4.4.1 Bound on the number of ships in the lock

Suppose the sizes of the ships are identical and the lock can accommodate at most b ships at once. It is easy to see that Lemma 3.2 and its proof also hold in this case. There exists an optimal strategy for which all upmoments and downmoments are contained in U and D, respectively. However, Lemma 3.1 is not directly applicable. Indeed, it can happen that ships need to wait longer than $4T$ when the capacity of the lock is filled. Suppose that during a certain time period no ships arrive at the lock. Then, the lock will go up and down with full capacity and without waiting until the waiting queue is empty. In other words, the strategy of the lock is very simple in this time period. Given that there are $n_1 + n_2$ ships in the instance, let $\eta = max\{n_1, n_2\}$. Then the following lemma holds:

Lemma 4.3 *When, for a given instance of the lockmaster's problem with capacity constraint, during a time period equal to $2T\lceil \frac{\eta}{b} \rceil$ no ships arrive at the lock, the instance can be divided into two instances. The solution can then be found by solving these two smaller instances.*

Proof. Suppose η ships are waiting at the lock to go up. Then the lock needs to go up and down until all ships are handled. Given that the lock has a capacity b, the queue will be empty after at most $\lceil \frac{\eta}{b} \rceil$ upmoments of the lock. $2T$ time units pass between two upmoments, such that the last ships go upstream at time $2T(\lceil \frac{\eta}{b} \rceil - 1)$. Note that the lock does not spend any

time waiting as the ships have already arrived and are waiting to move as soon as possible. Thus, T time units later the lock is at the upstream level, and another T time units later again at the downstream level. If during $2T\lceil\frac{\eta}{b}\rceil$ time units no ships arrive, the instance can be split into two separate instances. □

It follows that we can assume, without loss of generality, that each $2T\lceil\frac{\eta}{b}\rceil$ time units at least one ship and at most n ships arrive. In a $2T(\lceil\frac{\eta}{b}\rceil+1)$ interval there can be at most $O(n^2)$ elements in $U \cup D$ for that interval. We have at most n intervals, such that there are at most $O(n^3)$ elements in $U \cup D$.

Define a dynamic programming algorithm (DPc) with $f(u_i, d_j, p, q)$ (with $u_i \leq d_j - T$) as the minimal costs of a lockage strategy that includes the accumulated cost for all up-requests up to u_i and the cost for all down-requests up to d_j. Part of these ships is still waiting at the lock, i.e. p is the number of ships waiting to go upstream and q is the number of ships waiting to go downstream; the cost for these ships is only partial (indeed, their waiting time is not completed yet). This state features an upmoment at time $t = u_i$, a downmoment at time $t = d_j$, and there are no other up- or downmoments in between u_i and d_j. Let $l(u_{i'}, u_i)$ be equal to the number of ships i with arrival time s_i in the interval $(u_{i'}, u_i]$ and $k(d_{j'}, d_j)$ the number of ships j with arrival time r_j in the interval $(d_{j'}, d_j]$.
Then, let:

$$P = \begin{cases} \{max\{p + b - l(u_{i'}, u_i), 0\}\} & \text{if } p > 0 \\ \{0, 1, \ldots, b - l(u_{i'}, u_i)\} & \text{if } p = 0 \end{cases}$$

$$Q = \begin{cases} \{max\{q + b - k(d_{j'}, d_j), 0\}\} & \text{if } q > 0 \\ \{0, 1, \ldots, b - k(d_{j'}, d_j)\} & \text{if } q = 0 \end{cases}$$

and:

$$f(u_i, d_j, p, q) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T \\ p' \in P \\ q' \in Q}} \{f(u_{i'}, d_{j'}, p', q') + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k) + p'(u_i - u_{i'}) + q'(d_j - d_{j'})\}. \quad (1)$$

with initial state

$$f(u_1, u_1 + T, 0, 0) = 0$$

and $u_1 = \min\{s_1, r_1 - T\}$.
When $p, q > 0$ it means that the lock was operated at full capacity in the previous state. Just before operating the lock there were thus $p + b$ ships ready to go up, from which $l(u_{i'}, u_i)$ arrived between the previous upmoment of the lock and the current upmoment. Thus, after the previous upmoment of the lock there were $p + b - l(u_{i'}, u_i)$ ships not handled yet. If this is a negative number it means that all ships are handled up till $s_l \leq u_{i'}$ and $p' = 0$. When $p, q = 0$, it means that no ships are waiting and full capacity b was not necessarily used, meaning that $p' + l(u_{i'}, u_i) \leq b$. It follows that $p' \leq b + l(u_{i'}, u_i)$, and idem for q'. The waiting time of any ship l that arrived between u_i and $u_{i'}$ is at least $u_i - s_l$, which explains the second part of (1). However, for the p' ships that could not enter the lock at $u_{i'}$, the waiting time increases with $(u_i - u_{i'})$, which is dealt with in the third part. Analogue arguments hold for the downmoments. The optimal value is given by $\min\{f(u_i, d_j, 0, 0) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$.

Theorem 4.4 *DPc is a polynomial-time algorithm for the lockmaster's problem extended with a bound on the number of ships that fit together in the lock.*

Proof. The proof is analogue as the proof for Theorem 3.4. U and D have cardinality $O(n^3)$; p and q have cardinality $O(n)$. It follows that this algorithm will have $O(n^6)$ states. For each state we need to evaluate $O(n^6)$ states, yielding a total time complexity of $O(n^{12})$. This can be further reduced using similar arguments as in the proof of Theorem 3.4. □

Notice that when the ships are weighted, ships might no longer be locked in order of arrival and hence algorithm DP (or an extension of it) might fail to find an optimal solution. When considering the unidirectional case, Baptiste's algorithm [2] (see Section 2) yields a polynomial time procedure.

4.4.2 Bound on the size of the lock - no overtaking of ships allowed

Let us now consider the case where ships can have different sizes while the capacity of the lock is restricted. We will assume in this section that the ships are dealt with on a first-come, first-serve basis. Hence, no overtaking is possible. This is not unrealistic, see e.g. [17, 18]. We now state a DP for

this situation.
In a preprocessing step, the subsets of consecutive ships that fit together in the lock, are determined. Here, it is possible to include all kinds of filling and entering rules that can be important in practice, see e.g. [19]. This step can be executed in $O(n^2)$ time. Define the set A as the set containing all pairs (a, a'), with $s_a, s_{a'} \in S$, for which it holds that all downstream going ships arriving at $t \in [s_a, s_{a'}]$ fit together in the lock. Analogue, the set B is defined as the set containing all pairs (b, b'), with $r_b, r_{b'} \in R$, for which it holds that all upstream going ships arriving at $t \in [r_b, r_{b'}]$ fit together in the lock
There exists an optimal strategy for which up- and down-moments are contained in U and D, such that Lemma 3.2 holds. As in the previous section, we propose an alternative for Lemma 3.1. The size of the lock must be such that it can fit the largest ship in the instance. Using the same argument as in the previous section with $b = 1$ this means that when inter-arrival time is larger than $2nT$, the instance can be split. Now define a dynamic programming algorithm (DPc2) with $f(u_i, d_j, a, b)$ (with $u_i \leq d_j - T$, $u_i \geq s_a$, $d_j \geq r_b$) as the minimal cost for handling all up-requests up to r_a, all down-requests up to s_b, with the final up-moment at time u_i and the final down-moment at time d_j and no up- or down-moments in between. Then,

$$f(u_i, d_j, a, b) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T \\ (a', a) \in A \\ (b', b) \in B}} \{f(u_{i'}, d_{j'}, a', b') + \sum_{l: s_l \in [s_{a'}, s_{a-1}]} (u_i - s_l) + \sum_{k: r_k \in [r_{b'}, r_{b-1}]} (d_j - r_k)\}.$$

The optimal value is given by $\min\{f(u_i, d_j, n_1, n_2) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$.

Theorem 4.5 *DPc2 is a polynomial-time algorithm for the lockmaster's problem with a bound on the size of the lock and no overtaking of ships is allowed.*

Proof. The proof is analogue as the proof for Theorem 3.4. U and D have cardinality $O(n^3)$; a and b have cardinality $O(n)$. It follows that this algorithm will have $O(n^6)$ states. For each state we need to evaluate $O(n^6)$ states, yielding a total time complexity of $O(n^{12})$, which can be reduced as described before. □

4.5 Multiple (parallel) chamber lock

In practice a lock often consists in multiple chambers that operate independently such that ships can be dealt with in parallel. We will show that when the number of chambers is independent from the input and all chambers have identical lockage times, the problem can be solved in polynomial time by adapting DP. However, when the number of chambers is part of the instance and the chambers have arbitrary lockage times, the problem becomes NP-hard.

First, consider a problem with $k < n$ identical chambers in parallel, lockage time for all locks is equal to T. All possible lockage times are identical to the single chamber case, such that Lemma 3.2 and Lemma 3.1 are applicable. Indeed, each of the chambers will only move upon arrival of a ship or immediately after an up- (or down-) movement of the chamber. Let $\bar{u}_i$ be a vector of size k containing elements from U, thus $\forall l \leq k : \bar{u}_i(l) \in U$; and $\bar{d}_j$ a vector of size k containing elements from D, thus $\forall l \leq k : \bar{d}_j(l) \in D$.

Lemma 4.6 *Given that there are k uniform parallel chambers in the lockmaster's problem, an optimal solution exists where the lockage sequence of the chambers is ordered as follows $\bar{u}_i^*(1) < \bar{u}_i^*(2) < \ldots < \bar{u}_i^*(k)$ and $\bar{d}_j^*(1) < \bar{d}_j^*(2) < \ldots < \bar{d}_j^*(k)$, $\forall \bar{u}_i^*, \forall \bar{d}_j^*$.*

Proof. Suppose the optimal solution is not in accordance to Lemma 4.6. Then, there is a moment in time where the lockage sequence alters, let this moment be e.g. $\bar{d}_j(2) < \bar{d}_j(1)$. Given that $\bar{u}_i(1) < \bar{u}_i(2)$, it holds that chamber 1 is available to go down earlier than chamber 2. All chambers are identical, thus the solution value will not change when chamber 1 goes down at $t = \bar{d}_j(2)$ and chamber 2 at $t = \bar{d}_j(1)$, yielding a solution as described in Lemma 4.6. □

Let us now define $f(\bar{u}_i, \bar{d}_j)$ with $\bar{u}_i$ and $\bar{d}_j$ ordered and $\bar{u}_i(l) \leq \bar{d}_j(l) - T$, $\forall l \in 1 \ldots k, \bar{u}_i(l) \in U, \bar{d}_j(l) \in D$, as the minimal cost of a lockage strategy where all up requests up to $t = \bar{u}_i(k)$ and all down requests up to $t = \bar{d}_j(k)$ are dealt with. For each $l \in 1 \ldots k$, chamber l moves up at time $t = \bar{u}_i(l)$ and down at time $t = \bar{d}_j(k)$ and there are no other up- or down-moments in between.

Then, for all $\bar{u}_i$ and $\bar{d}_j$, $\bar{u}_i(l) \leq \bar{d}_j(l) - T$ we have

$$f(\bar{u}_i, \bar{d}_j) = \min_{\substack{\bar{u}'_i(k)<\bar{u}_i(1) \\ \bar{d}'_j(k)<\bar{d}_j(1)}} \{f(\bar{u}'_i, \bar{d}'_j)+$$

$$\sum_{l=0\ldots k-1} \sum_{\substack{m: \\ s_m \in (\bar{u}_i(l), \bar{u}_i(l+1)]}} (\bar{u}_i(l) - s_m) + \sum_{l=0\ldots k-1} \sum_{\substack{o: \\ r_o \in (\bar{d}_j(l), \bar{d}_j(l+1)]}} (\bar{d}_j(l) - r_o)\},$$

with $\bar{u}_i(0) = \bar{u}'_i(k)$ and $\bar{d}_j(0) = \bar{d}'_j(k)$. The optimal value is given by $\min\{f(\bar{u}_i, \bar{d}_j) | \bar{u}_i(k) \geq s_{n_2}, \bar{d}_j(k) \geq r_{n_1}, \bar{u}_i(l) \in U, \bar{d}_j(l) \in D, \forall l \leq k\}$.

Lemma 4.7 *The lockmaster's problem with multiple identical parallel chambers is solvable in polynomial time.*

Proof. See also the proof of Theorem 3.4. There are $O(n^{3k})$ states, computing each state can be done by evaluating $O(n^{3k})$ states, leading to a total time complexity for the algorithm equal to $O(n^{6k})$. □

5 Non-identical parallel chambers

In this section we prove that in the case of multiple non-identical parallel chambers where the number of chambers is part of the input, the lockmaster's problem is NP-hard.

Lemma 5.1 *The lockmaster's problem with non-identical parallel chambers is strongly NP-hard.*

Proof. We show that the lockmaster's problem with multiple non-identical parallel chambers is at least as hard as numerical matching with target sums (NMTS). In an instance of NMTS we are given positive integers a_i $(1 \leq i \leq n)$, b_j $(1 \leq j \leq n)$ and t_κ $(1 \leq \kappa \leq n)$. It holds that $\sum_\kappa t_\kappa = \sum_{i,j}(a_i + b_j)$. The question is whether there exists a collection of m triples (i, j, κ) such that (i) $a_i + b_j = t_\kappa$ for each triple, and (ii) each integer in the input occurs exactly once. This problem is proven to be NP-hard by Garey and Johnson [11]. We assume, without loss of generality, that the a_i's and t_κ's are pairwise different and that $\min_j b_j > \max_i a_i$. We now construct an instance of the lockmaster's problem as follows. There are $2n$ ships, n ships travel upstream and arrive at the lock at $s_l := a_i$ and n ships travel downstream arriving at the lock at times $r_k := t_\kappa$. There are n chambers, each with a certain lockage time b_j. Is there a solution for the lockmaster's

problem with total waiting time equal to 0? If there is a solution to NMTS, each triple (a_i, b_j, t_κ) corresponds to a combination of a chamber with an upstream and a downstream going ship. The upstream going ship arrives at time a_i, enters the chamber that needs b_j time units to arrive at the downstream level and after t_κ time units the downstream going ship enters the chamber and spends b_j time units in the lock. Each ship can enter a chamber upon arrival time and total waiting time is equal to 0. On the other hand, if a solution to the lockmaster's problem with value 0 exists, it means that each upstream going ship is assigned upon arrival to exactly one chamber. Moreover, since $\min_j b_j > \max_i a_i$, it follows that each chamber accommodates one upstream going ship. Since downstream going ships also have waiting time equal to 0, there must exist triples for which it holds that $a_i + b_j = t_\kappa$ and we have a solution to NMTS. □

6 Further research

In this work we study the lockmaster's problem for a single lock, with a single or with multiple chambers. A relevant question is how to deal with the problem when there are multiple locks in series, either with ships arriving from upstream at the first lock and downstream at the last lock, or, more complex, with ships also arriving at intermediate locks. In general, more complex waterway networks with several locks would be a nice subject for future work. In reality, lockmasters do not know in advance all the arrival times of the ships, only from ships that are already at a certain distance from the lock. Studying the online version of the lockmaster's problem could capture this element in a better way. A different direction for furture research is to go further into the batch scheduling aspect of the problem, e.g. what happens if there are three or more job families instead of the two studied in this paper.

Acknowledgements We like to thank Cor Hurkens for an interesting discussion on the subject.
Dr. S. Coene is a post-doctoral research fellow of the "Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO)".

Appendix

In this appendix we give a proof that shows the correctness of the DP from Section 3.

We have defined a dynamic programming algorithm (DP) where $f(u_i, d_j)$ (with $u_i \leq d_j - T$) represents the minimal costs of a lockage strategy that takes care of all up-requests up to u_i, all down-requests up to d_j, which features an upmoment at time $t = u_i$, which features a downmoment at time $t = d_j$, and such that there are no other up- or downmoments in between u_i and d_j. We claim that these costs can be computed recursively as follows.

For each $u_i \in U$, $d_j \in D$, with $u_i \leq d_j - T$ we have:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\};$$

for all $u_i > d_j - T$ we set:

$$f(u_i, d_j) = \infty.$$

With $f(u_1, u_1 + T) = 0$ and $u_1 = \min\{s_1, r_1 - T\}$. The optimal value is given by $\min\{f(u_i, d_j) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$.

Proof. We will show that the values of $f(u_i, d_j)$ computed by DP satisfy the definition given above. First, observe that Lemma's 3.1 and 3.2 imply that we can restrict ourselves to considering states $(u_i, d_j) \in U \times D$. Now, suppose there exist states $(u_i, d_j) \in U \times D$, for which the cost computed according to the recursion does not correspond to the definition. Then there is a "smallest" such state, i.e. we consider that state with minimal d_j (d^*), and next with minimal u_i (u^*). The value of that state is :

$$f(u^*, d^*) = \min_{\substack{d_{j'} \leq u^* - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u^*]} (u^* - s_\ell) + \sum_{k : r_k \in (d_{j'}, d^*]} (d^* - r_k)\}.$$

Consider the right-hand side. Notice that we know, by minimality of (u^*, d^*), that the term $f(u_{i'}, d_{j'})$ satisfies the definition. Thus, the right-hand side considers all possible feasible strategies that take care of all up-requests up to $u_{i'}$, all down-requests up to $d_{j'}$, and it assigns to $f(u^*, d^*)$ the value of the strategy with minimum cost. Hence, it is correct, and we have arrived at a contradiction. □

References

[1] H. Allaeys. Optimalisering van een sluis (in Dutch), 2010. Master Thesis, Katholieke Universiteit Leuven.

[2] P. Baptiste. Batching identical jobs. *Mathematical Methods of Operations Research*, 52:355–367, 2000.

[3] M. Boudhar. Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, 57:513–527, 2003.

[4] P. Brucker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. Van De Velde. Scheduling a batching machine. *Journal of Scheduling*, 1:31–54, 1998.

[5] Waterborne Commerce Statistics Center. Waterborne commerce from the united states, September 2011. `http://www.ndc.iwr.usace.army.mil/wcsc/pdf/wcusnatl09.pdf`, `accessed 12.09.2011`.

[6] T. C. E. Cheng, J. J. Yuan, and A. F. Yang. Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers and Operations research*, 32:849–859, 2005.

[7] European commission. Promotion of inland waterway transport, January 2011. `http://ec.europa.eu/transport/inland/promotion/promotion-en.htm`.

[8] A. Condotta, S. Knust, and N. V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13:463–477, 2010.

[9] Inland Navigation Europe. Water webletter, November 2010. `www.inlandnavigation.org`.

[10] G. Finke, V. Jost, M. Queyranne, and A. Sebő. Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156:556–568, 2008.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[12] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10:256–269, 1981.

[13] C. Lee, R. Uzsoy, and L. A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40:764–775, 1992.

[14] R. M. Nauss. Optimal sequencing in the presence of setup times for tow/barge traffic through a river lock. *European Journal of Operational Research*, 187:1268–1281, 2008.

[15] C. T. Ng, T. C. E. Cheng, J. J. Yuan, and Z. H. Liu. On the single machine serial batching scheduling problem to minimize total completion time with precedence constraints, release dates and identical processing times. *Operations Research Letters*, 31:323–326, 2003.

[16] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[17] L. D. Smith, D. C. Sweeney, and J. F. Campbell. Simulation of alternative approaches to relieving congestion at locks in a river transportation system. *Journal of the Operational Research Society*, 60:519–533, 2009.

[18] C. Ting and P. Schonfeld. Control alternatives at a waterway lock. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 127:89–96, 2001.

[19] J. Verstichel and G. Vanden Berghe. A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, 5:457–478, 2009.

[20] S. Webster and K. R. Baker. Scheduling groups of jobs on a single machine. *Operations Research*, 43:692–703, 1995.